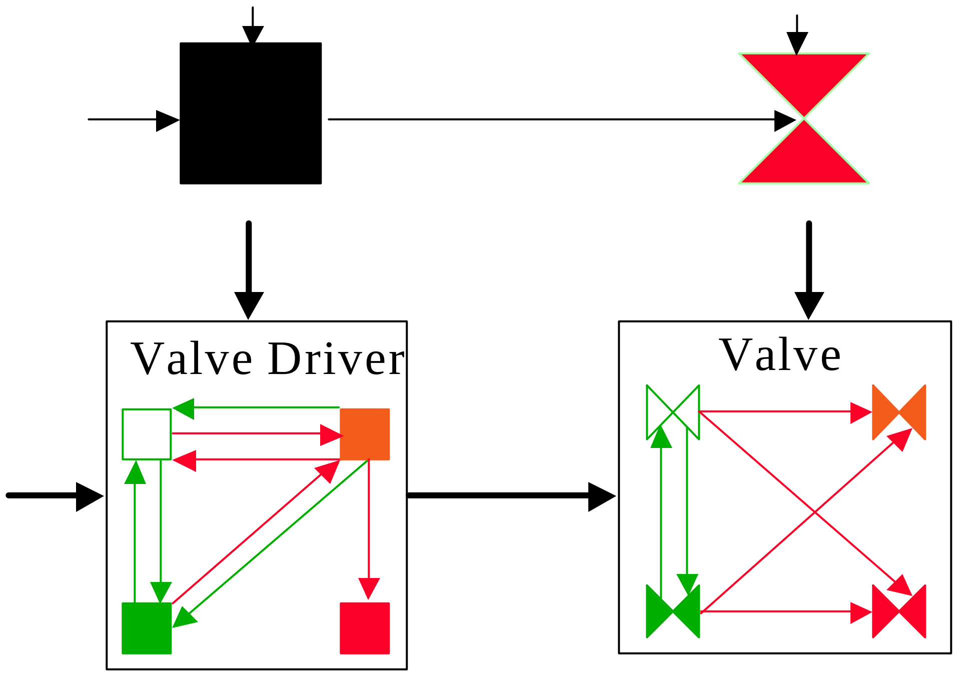
Model-based Reactive Planning
How does it relate to STRIPS planning?

STRIPS: **Planning:** **IF** clear(top) and on(A,B)
THEN Delete on (A,B) and clear(top)
Add on(top,B)

Model-based Reactive Planning:



Partially observability
exogenous effects
indirect control
concurrent

Comparing MRP and STRIPS

STRIPS Planning

- action representation
  - strips operators with precondition and add/delete as effects.
- state variables only change directly by operator add/delete.
- Operators are invoked directly
- State is held constant when operators are not invoked.
- Operators are invoked one at a time.

Model-based Reactive Planning

- action representation
  - state transitions ρ_τ
  - co-temporal interactions ρ_Σ
- state variables change through transitions or through interactions.
- Transitions are controlled by establishing control values which interact with internal variables.
- State changes may not be preventable.
- Enabling one transition may necessarily cause a second transition to occur.

How Burton Achieves Reactivity

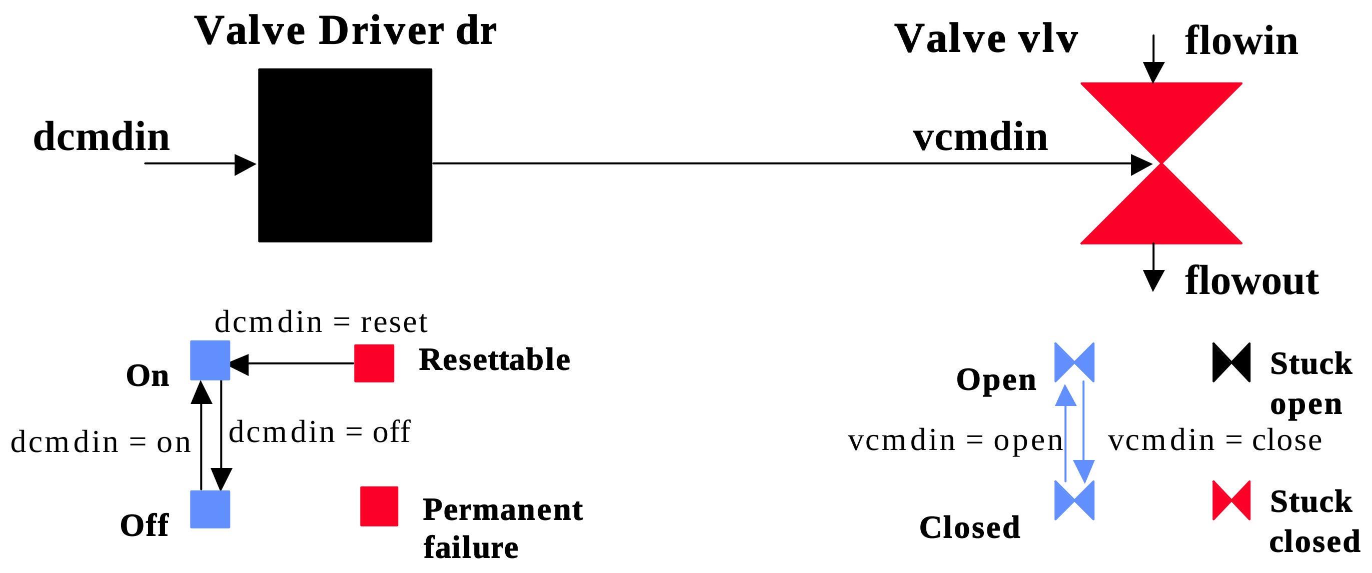
Problem: Model-based Planning is NP Hard.
Solution:

- Model compilation eliminates cotemporal interactions ρ_Σ, hence presolving NP hard part while preserving expressivity.
- Exploit fact that hardware typically behaves like STRIPS ops.
  - individual controllability & persistance
- Exploit requirement that the planner avoid damaging effect.
- Exploit causal, loop-free structure of hardware topology.
- Compile transitions into a compact set of concurrent policies.

Burton Model-based Reactive Planner: [Williams & Nayak 97]

Generates first plan action in average case constant time.

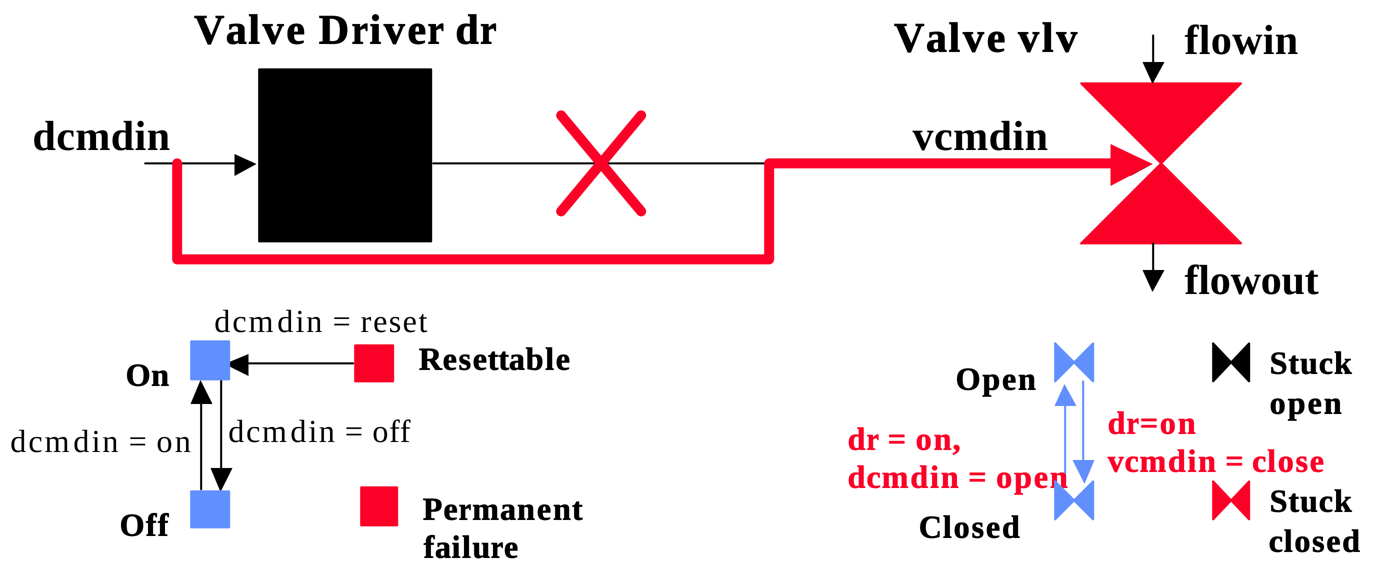
Driver Valve Example



- dr = resettable & dcmdin = reset $\Rightarrow$ next (dr1 = on)
- dr1 = on & dcmdin = open $\Rightarrow$ vcmdin = open
- . . .

- vlv = closed & vcmdin = open $\Rightarrow$ next (vlv = open)
- vlv = open & flowin = pos $\Rightarrow$ flowout = pos
- . . .

1. Model Compilation



Idea:

Eliminate hidden variables (vcmdin) and cotemporal interactions ρ_{Σ} , resulting in transitions that depend only on control variables (dcmdin) and state variables (dr,vlv).

Models are Compiled through Prime Implicate Generation

- Compiled transitions are all formula of the form

$$\Phi_i \Rightarrow \text{next}(y_i = e_i)$$

implied by the original transition specification, where Φ_i is a smallest conjunction without hidden variables (i.e., prime implicates).

- Example:

vlv = closed & vcmdin = open ⇒ next (vlv = open)

dr1 = on & dcmdin = open ⇒ vcmdin = open

compile to:

vlv = closed & dr = on & dcmdin = open ⇒ next (vlv = open)

- 40 seconds on SPARC 20 for 12,000 clause spacecraft model.

Simplifying to Strips

- Difference 1: Transitions can occur without control actions.
  - tub = empty & faucet = on ⇒ next (tub = non-empty)
- Requirement 1:
  - Each control variable has an idling assignment.
  - No idling assignment appears in any transition.
  - The antecedent of every transition includes a non-idling control assignment.
- Example:
  - drcmdin has idling value “none” and non-idling dcmdin = open
  - vlv = closed & dr = on & dcmdin = open ⇒ next (vlv = open)

Simplifying to Strips (cont.)

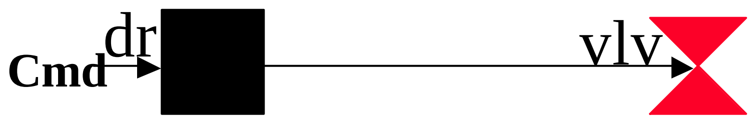
- Difference 2: Control actions can invoke multiple transitions.
  - vlv1 = closed & dr = on & dcmdin = open $\Rightarrow$ next (vlv1 = open)
  - vlv2 = closed & dr = on & dcmdin = open $\Rightarrow$ next (vlv2 = open)
- Definition: The control(state) conditions of a transition are the control(state) variable assignments of its antecedent condition.
  - state condition: vlv1 = closed & dr = on
  - control condition: dcmdin = open
- Requirement 2:
  - No set of control conditions of one transition is a proper subset of the control conditions of a different transition.

But STRIPS is still intractable.

Reasons Search is Needed

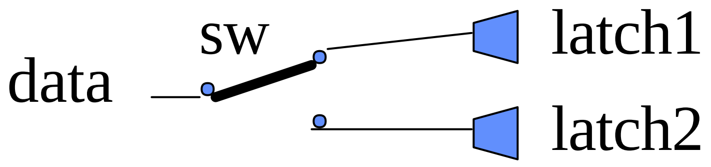
1) An achieved goal can be clobbered by a subsequent goal.

- e.g., achieving dr = off and then vlv = open clobbers dr = off.



2) Two goals can compete for the same variable in their subgoals.

- e.g., latch1 and latch2 compete for the position of switch sw.



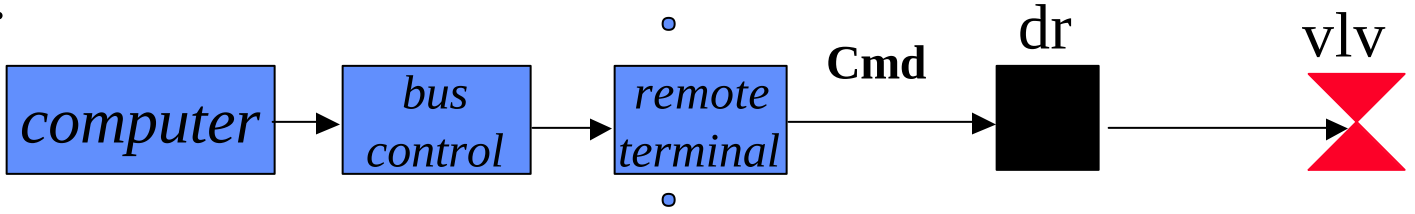
3) A state transition of a subgoal variable has irreversible effect.

- e.g., assume sw can be used once, then latch1 must be latched before latch2.

➡ *To achieve reactivity we eliminate all forms of search.*

Exploiting Causality to Avoid Threats

- Observation: Component schematics tend not to have feedback loops.



- The *Causal Graph* G of compiled transition systems S is a directed graph whose vertices are state variables. G contains an edge from v1 to v2 if v1 occurs in the antecedent of v2's transition.

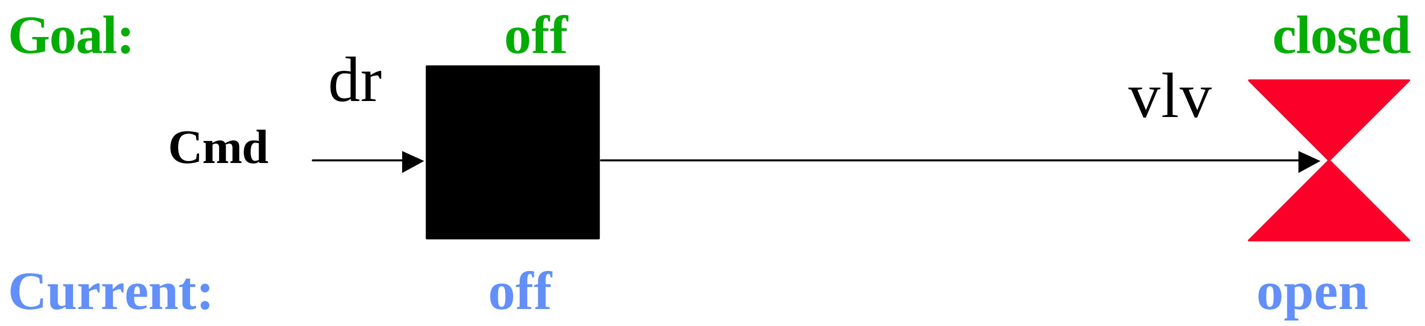
$$\begin{tikzcd} & dr \arrow[r] & vlv \\ dcmdin \arrow[ur] \arrow[urr] & & \end{tikzcd}$$

Requirement 3: The causal graph must be acyclic.

➡ *How can this causality be exploited?*

Exploiting Causality to Avoid Threats

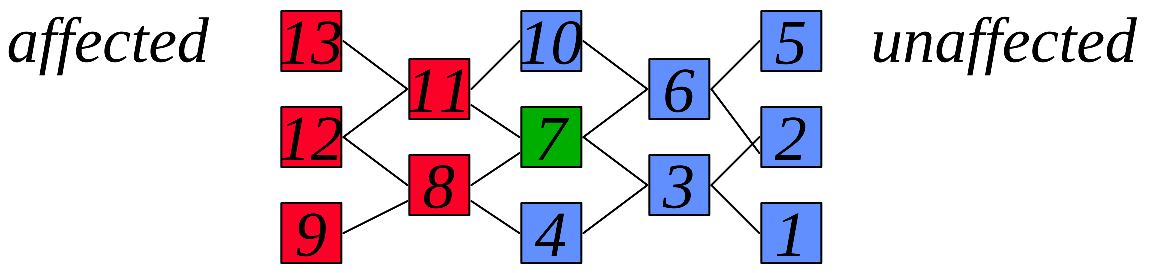
Idea: Achieve goals by working from effects to causes (e.g., vlv then dr), completing one goal before starting the next.



- work on vlv = closed
  - work on dr = on
    - next-action: Cmd = dr-on
  - next action: Cmd = vlv-close
- work on dr = off
  - next action: Cmd = dr-off

How to Avoid Clobbering Sibling Goals

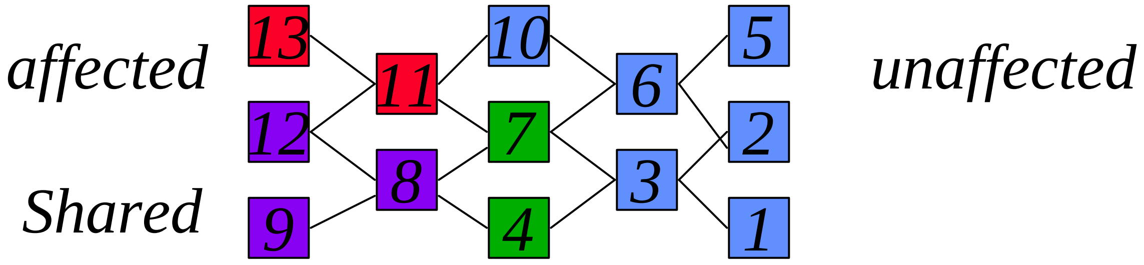
- The only variables necessary to achieve y = e are the ancestors of y, y can be changed without affecting its descendants.



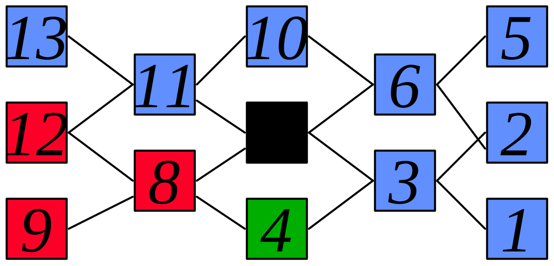
- To avoid clobbering achieved goals Burton solves goals in an upstream order.
- Upstream order corresponds to achieving goals in order of increasing depth first number.

How to Avoid Clobbering Shared Subgoals

- Shared ancestors of sibling goals are required to establish both goals.



- Ancestors are no longer needed once goal has been satisfied.



- **Solution:** To avoid clobbering shared subgoal variables, solve one goal before starting on next sibling.

→ **Generates first control action first!**

Burton: Online Algorithm (incomplete)

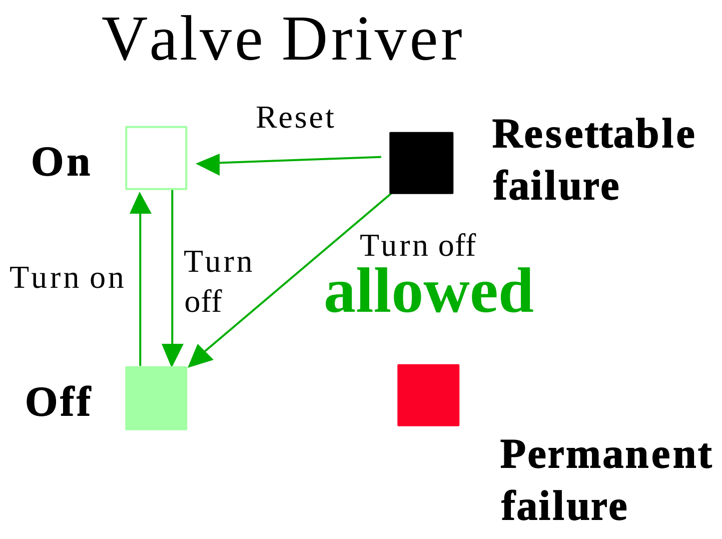
NextAction(initial state θ, target state γ, compiled system S')

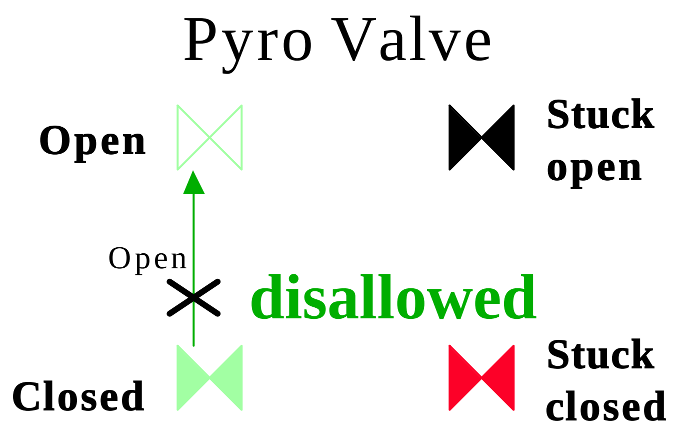
- **Select unachieved goal:** Find unachieved goal assignment with the lowest topological number. If all achieved return **Success.**
- **Select next transition:** Let t_y be the transition graph in S for goal variable y. Nondeterministically select a path p along transitions in t_y from e_i to e_f. Let SC and CC be the state and control conditions of the first transition along p.
- **Enable transition:** Control = NextAction(θ,SC,S'). If Control = **Success** then state conditions SC are already satisfied, return CC to effect transition. If **Failure** return it. Otherwise Control contains control assignments to progress on SC. Return Control.

⇨ Some *search still remains*

Exploiting Safety

- Requirement 4: Only reversible transitions are allowed, except when repairing a component.



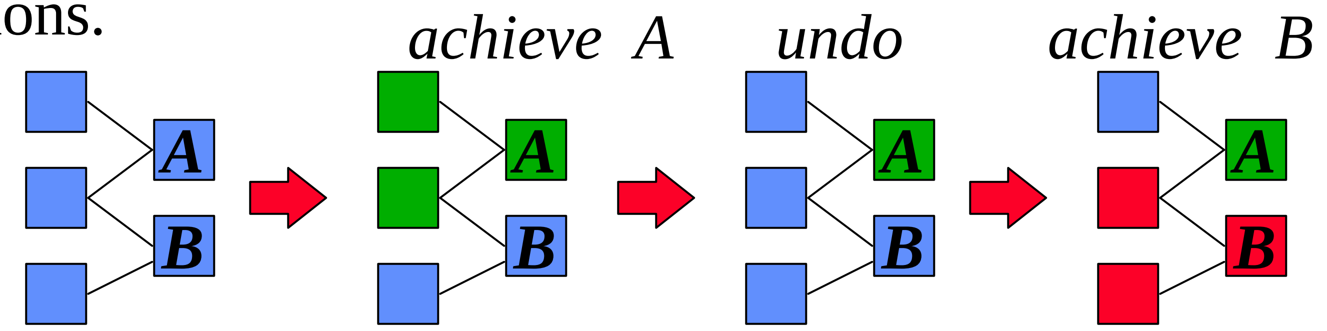


Rationale: Irreversible actions expend non-renewable resources.
Should only be performed after careful (human?) deliberation.

Using Reversibility to Avoid Deadend (Sub) Goals

Lemma:

- A & B is reachable from θ by reversible transitions exactly when A and B are separately reachable from θ by reversible transitions.



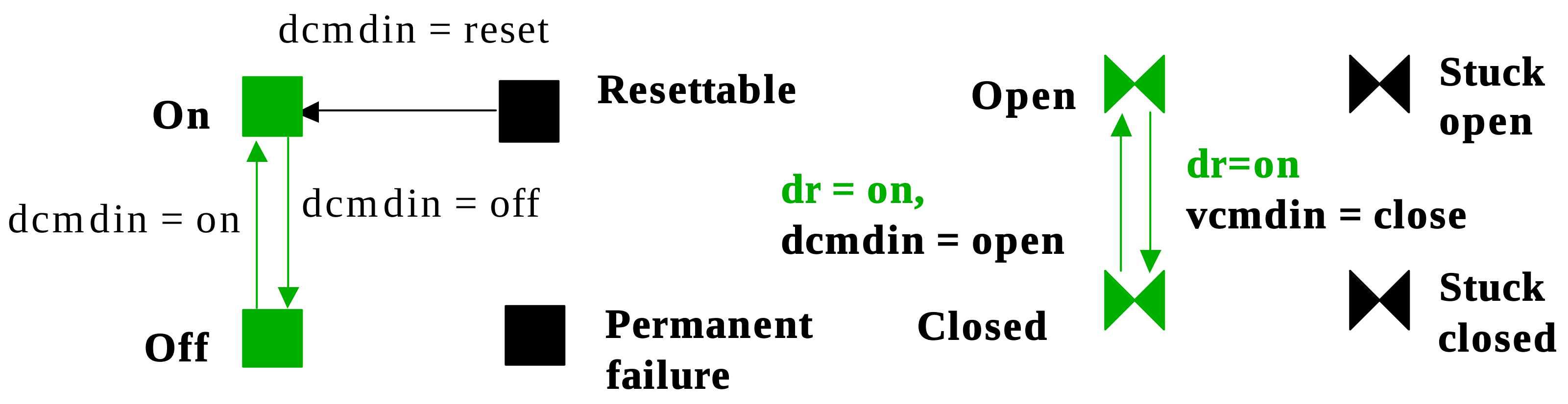
Idea:

- Precompute and label all assignments that can be **reversibly** achieved from initial state θ.
- Only use assignments labeled **reversible** as (sub)goal, and transitions involving **reversible** assignments.
- Exploit Lemma to test if top-level goals are achievable.

Defining Reversibility

Definition:

- An assignment y = e_k can be **Reversibly** achieved starting at y = e_i if there exists a path along **Allowed** transitions from initial value e_i to e_k and back.
- A transition is **Allowed** if all its state conditions are **Reversible**.

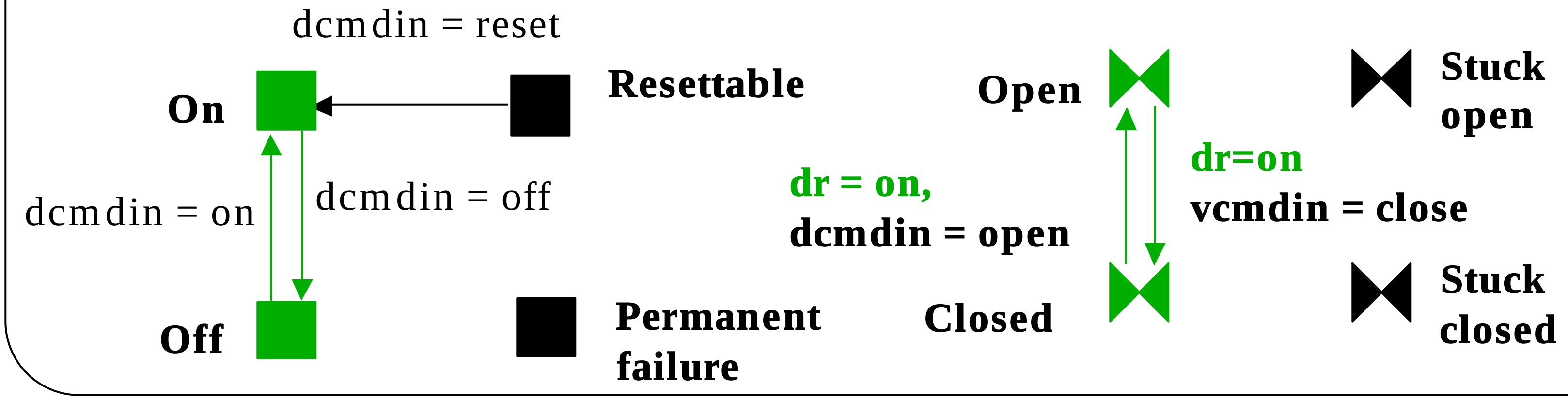


Burton: Reversibility Labeling Algorithm

LabelSystem(initial state θ, compiled system S')

For each state variable y of S' in decreasing topological order:

- For each transition τ_y of y, label τ_y **Allowed** if all its state conditions are labeled **Reversible**.
- Compute the strongly connected components (SCCs) of the **Allowed** transitions of y.
- Find y's initial value $y = e_i$ in θ. Label each assignment in the SCC of $y = e_i$ as **Reversible**.



Burton: Online Algorithm

NextAction(initial state θ, target state γ, compiled system S’,true?)

- **Solvable goals?:** When **top? = True,** unless each goal g in γ is labeled **Reversible**, return **Failure.**
- **Select unachieved goal:** Find unachieved goal assignment with the lowest topological number. If all achieved return **Success.**
- **Select next transition:** Let t_y be the transition graph in S for goal variable y. Find a path p in t_y from e_i to e_f along transitions labeled **Allowed**. Let SC and CC be the state and control conditions of the first transition along p.
- **Enable transition:** Control = NextAction(θ,SC,S’). If Control = **Success** then state conditions SC are already satisfied, return CC to effect transition. Otherwise Control contains control assignments to progress on SC. Return Control.

Incorporating Repair Actions

Definition: A repair is a transition from a failure assignment to a nominal assignment.

Idea:

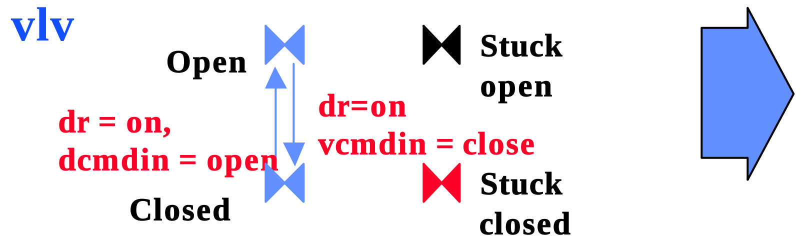
- Burton never uses a failure assignment to achieve a goal if the failure is repairable.
- Repair minimizes irreversible effects. If y is assigned failure e_{f}, Burton traverses allowed transitions from e_f to the first nominal assignment reached (nominal SCC w lowest number).
- If a failure assignment is not repairable then it can be used.

Eliminating Cost of Finding Transition Paths: Generating Concurrent Policies

- NextAction is O(e*m) where
  - e is the number of transitions for a single variable y.
  - m is the maximum depth in the causal graph.

⇒ table lookup

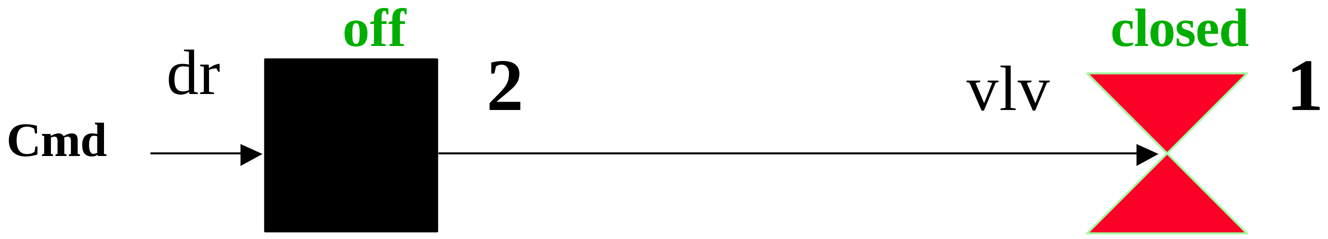
- Compute a feasible policy $\pi_y(e_i,e_f)$ for variable y, where
  - e_i is a current assignment
  - e_f is a goal assignment
  - $\pi_y(e_i,e_f)$ returns the sorted conditions of the first transition along a path from e_i to e_f.



Current \ Goal	open	closed
open	Idle	dr = on dcmdin=close
closed	dr = on dcmdin=open	Idle
stuck	Failure	Failure

Burton computes next action (step 1)

Goal:



Current: off open

Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed

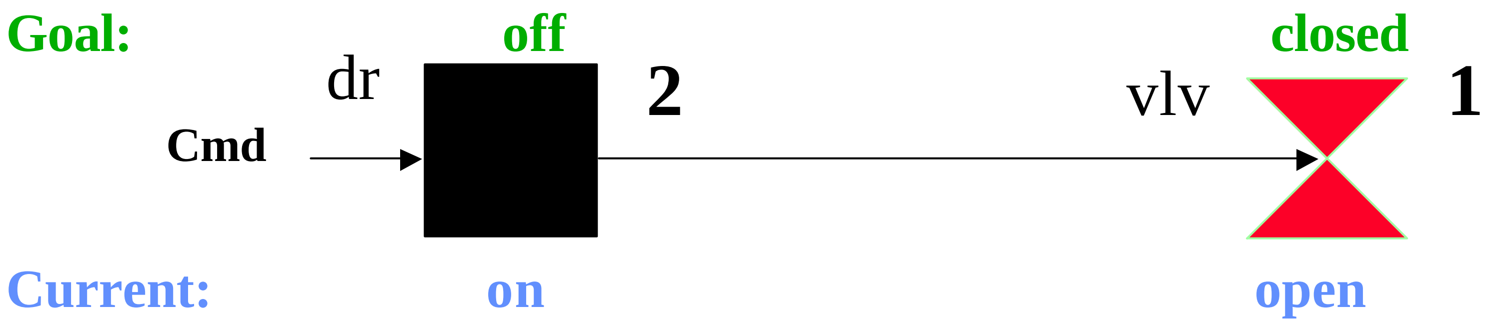
open	idle	dr = on
		dcmdin=close

closed	dr = on
	dcmdin=open
		idle

stuck	fail	fail

➡ dcmdin= on

Burton computes next action (step 2)



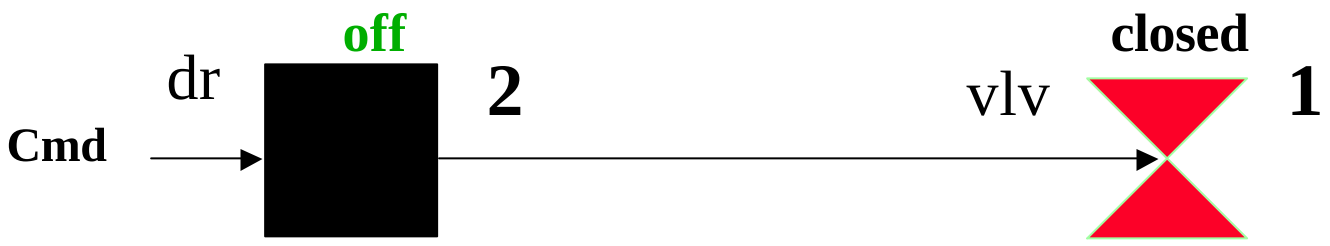
Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

→ **cmdin = close**

Failure occurs during plan execution Burton computes next action (step 3)

Goal:



Current: on -> reset failure closed

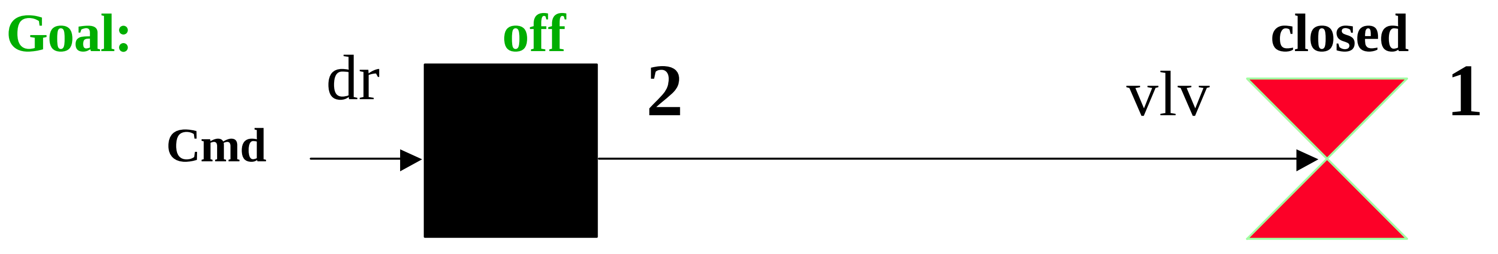
Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

→ cmdin = reset

Burton computes next action (step 4) completing plan

Goal:



Current: on closed

Current \ Goal	on	off
on	idle	dcmdin = off
off	dcmdin = on	idle
reset failure	dcmdin = reset	dcmdin = reset

Current \ Goal	open	closed
open	idle	dr = on , cmdin = close
closed	dr = on, cmdin = open	idle
stuck	fail	fail

→ cmdin = off

Burton Complexity: ConstantAverage Cost

Cost of generating the first action:

- Worst Case: Maximum depth of causal graph.
- Average Cost: Constant time.
  - Each edge of the goal/subgoal tree traversed twice.
  - Each node of the goal/subgoal tree generates one action.
  - # edges < 2 * # nodes.